python for Computational Problem Solving - pCPS - Lists Lecture Slides - Class #15_#16

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

pCPS Assignment Batches

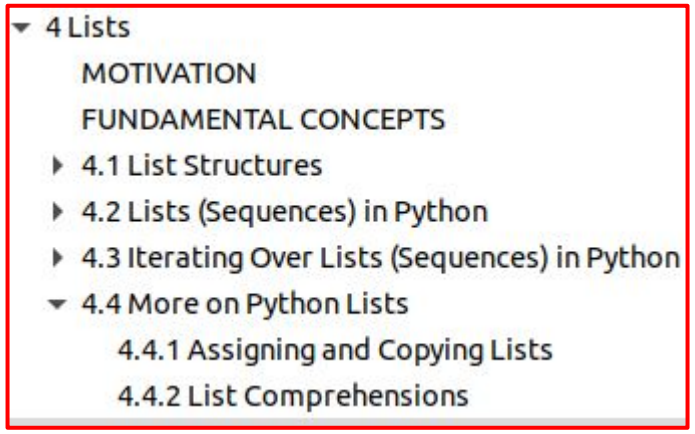
```
,BatchId,ProjectBatch
0,pCPS_Assignment_Batch_ID_1,"('PES1202100893', 'PES1202100956', 'PES1202101345')"
1,pCPS_Assignment_Batch_ID_2,"('PES1202100862', 'PES1202101351', 'PES1202100999')"
2,pCPS_Assignment_Batch_ID_3,"('PES1202100802', 'PES1202100895', 'PES1202101314')"
3,pCPS_Assignment_Batch_ID_4,"('PES1202101342', 'PES2202100686', 'PES2202100705 ')"
4,pCPS_Assignment_Batch_ID_5,"('PES1202100868', 'PES1202100891', 'PES1202101354')"
5,pCPS_Assignment_Batch_ID_6,"('PES1202100884', 'PES1202100886', 'PES1202101033')"
6,pCPS_Assignment_Batch_ID_7,"('PES1202101027', 'PES1202101339', 'PES1202101054')"
7,pCPS_Assignment_Batch_ID_8,"('PES1202100959', 'PES1202100991', 'PES1202101048')"
8,pCPS_Assignment_Batch_ID_9,"('PES1202101466', 'PES1202101481', 'PES1202100838')"
9,pCPS_Assignment_Batch_ID_10,"('PES1202101050', 'PES1202101415', 'PES1202100970')"
10,pCPS_Assignment_Batch_ID_11,"('PES1202100960', 'PES1202100860', 'PES1202100967')"
11,pCPS_Assignment_Batch_ID_12,"('PES1202100974', 'PES1202100877', 'PES1202101330')"
12,pCPS_Assignment_Batch_ID_13,"('PES1202100801', 'PES1202101349', 'PES1202101480')"
13,pCPS_Assignment_Batch_ID_14,"('PES1202100803', 'PES1202101020', 'PES1202101513')"
14,pCPS_Assignment_Batch_ID_15,"('PES1202101315', 'PES1202101458', 'PES1202101460')"
15,pCPS_Assignment_Batch_ID_16,"('PES2202100680', 'PES1202100836', 'PES1202101014')"
16,pCPS_Assignment_Batch_ID_17,"('PES2202100695', 'PES1202101416', 'PES1202100930')"
17,pCPS_Assignment_Batch_ID_18,"('PES1202100816', 'PES1202101407', 'PES1202100890')"
18,pCPS_Assignment_Batch_ID_19,"('PES1202100829', 'PES1202101353', 'PES1202100841')"
19,pCPS_Assignment_Batch_ID_20,"('PES1202100789', 'PES1202101306', 'PES1202100830')"
20,pCPS_Assignment_Batch_ID_21,"('PES1202101329', 'PES1202100807', 'PES1202101038')"
21,pCPS_Assignment_Batch_ID_22,"('PES1202101041', 'PES1202100835', 'PES1202101051 ')"
22,pCPS_Assignment_Batch_ID_23,"('PES2202100627', 'PES1202100864', 'PES1202101358')"
23,pCPS_Assignment_Batch_ID_24,"('PES1202100928', 'PES1202101522', 'PES1202100953')"
24.pCPS_Assignment_Batch_ID_25,"('PES1202101538', 'PES1202101325')"
```

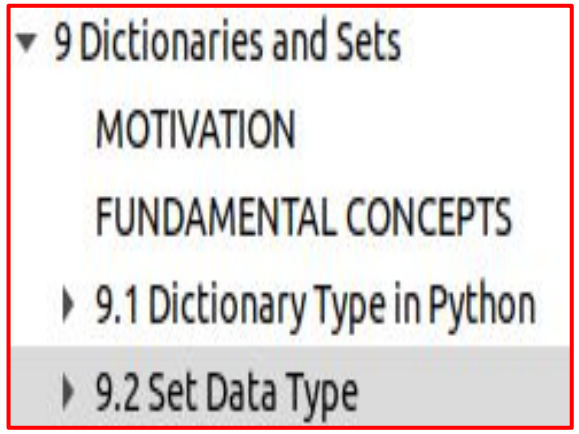
python for Computational Problem Solving Syllabus

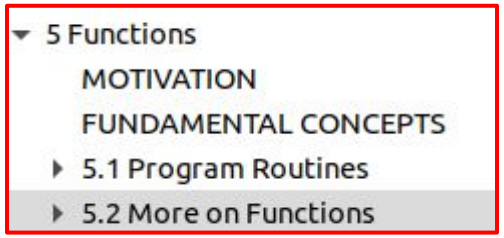
Unit II: Collections & Basics of Functions - 12 Hours

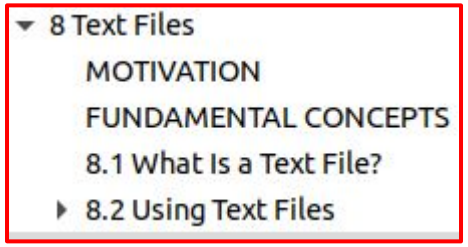
Lists, Tuples , Dictionaries, Sets, Strings and text file manipulation: reading and writing files. Functions : Definition, call.

T1: 4.1 – 4.4 - Class #15, #16, #17, #18
T1: 9.1 – 9.2 - Class #19, #20, #21, #22
T1: 5.1-5.2 - Class #23, #24
T1: 8.1, 8.2, 8.3 - Class #25, #26

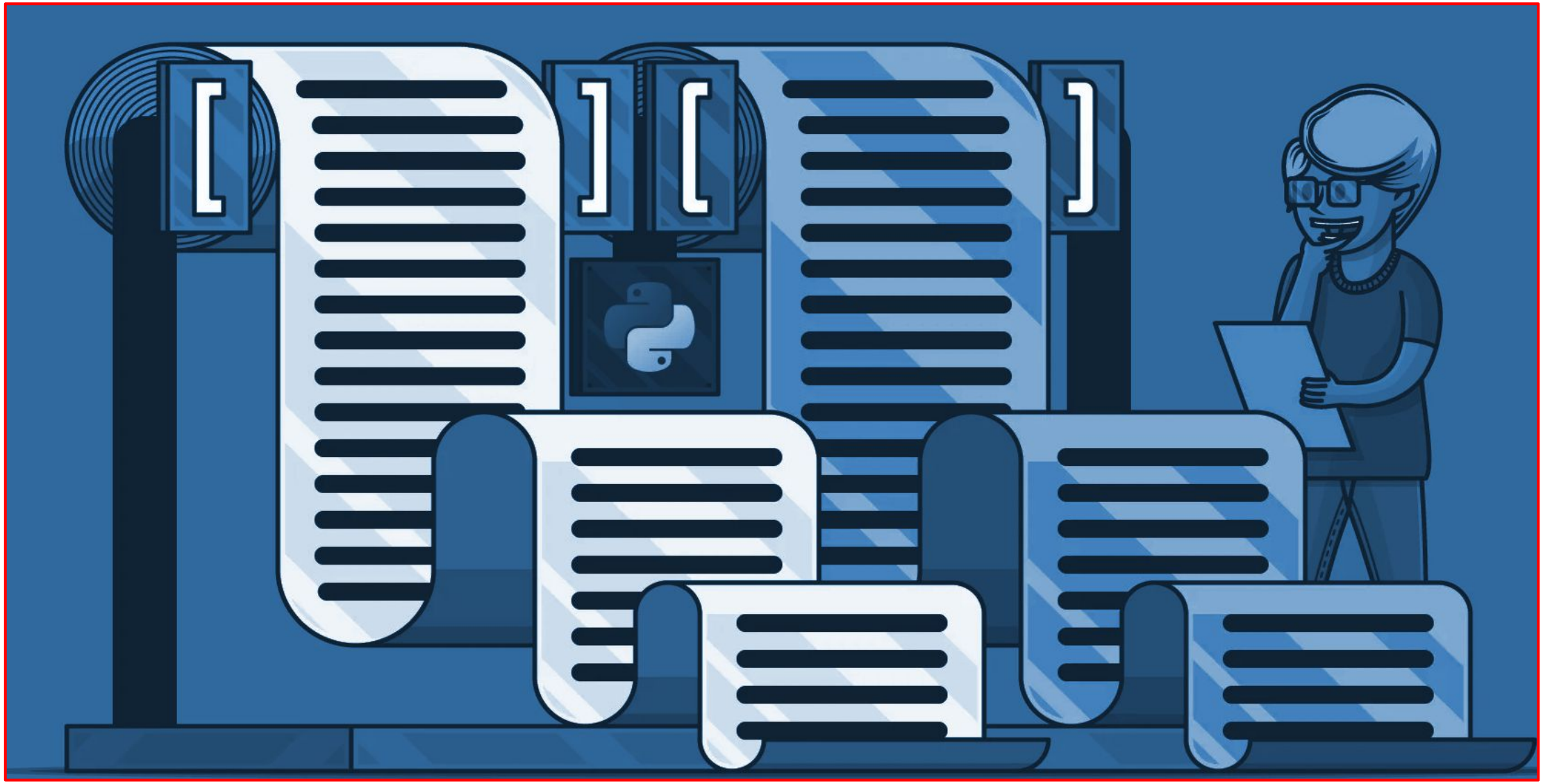








pCPS 4 Lists



pCPS 4.1 Collection Data Types

- python programming language has four collection data types
    - list
    - tuple
    - sets
    - dictionary.
- python also comes with a built-in module known as collections , which has specialized data structures which basically covers for the shortcomings of the four data types.
- With a python list, you can group python objects together in a one-dimensional row that allows objects to be accessed by position, added, removed, sorted, and subdivided
- The biggest reason to use a list is to able to find objects by their position in the list.

pCPS 4.1 Lists

- A **List** is the most basic python **Data Structure**.
- A **data structure** is a **specialized** format for **organizing**, **processing**, **retrieving** and **storing data**.
- **Data structures** make it **easy** for **users** to access and work with the data they need in appropriate ways.
- **Lists** can be list of objects or values.
- The **concept** of a **list** is similar to our **everyday notion** of a list.
- We read off (access) items on our **to-do list**, **add items**, **cross off** (delete) items, and so forth.
- To hold a sequence of values, python provides the **'list'** class

pCPS 4.1.1 What is a List ?

- A **list** is a linear data structure , meaning that its elements have a **linear ordering**.
- **Linear ordering** means that, there will be a **first** element, a **second** element, and so on
- It is customary in programming languages including **python** to begin **numbering sequences** of items with an **index** value of **0** rather than **1**.
- This is referred to as **zero-based indexing** .
- The **negative indexing** is the act of indexing from the **end** of the list with indexing starting at **-1** i.e. -1 gives the **last** element of list, **-2** gives the **second last** element of list and **so on**

pCPS 4.1.1 What Is a List ?

pCPS 4.1.1 What Is a List ?

```
# List of integer Elements
MyListIntegers = [1,2,3,4,5,0,-1,-2,-3,-4]
print(type(MyListIntegers))
print(MyListIntegers)
print('\n-------------------------------------')
print(MyListIntegers[4])
print(type(MyListIntegers[4]))
```

```
<class 'list'>
[1, 2, 3, 4, 5, 0, -1, -2, -3, -4]

-------------------------------------
5
<class 'int'>
```

```
# List of float Elements
MyListFloat = [1.5,2.5,3.5,4.5]
print(type(MyListFloat))
print(MyListFloat)
print('-------------------------------------')
Length = len(MyListFloat)
print('Length =', Length)
print(MyListFloat[Length-1])
print(MyListFloat[-Length])
print(type(MyListFloat[-Length]))
```

```
<class 'list'>
[1.5, 2.5, 3.5, 4.5]
-------------------------------------
Length = 4
4.5
1.5
<class 'float'>
```

pCPS 4.1.1 What Is a List ?

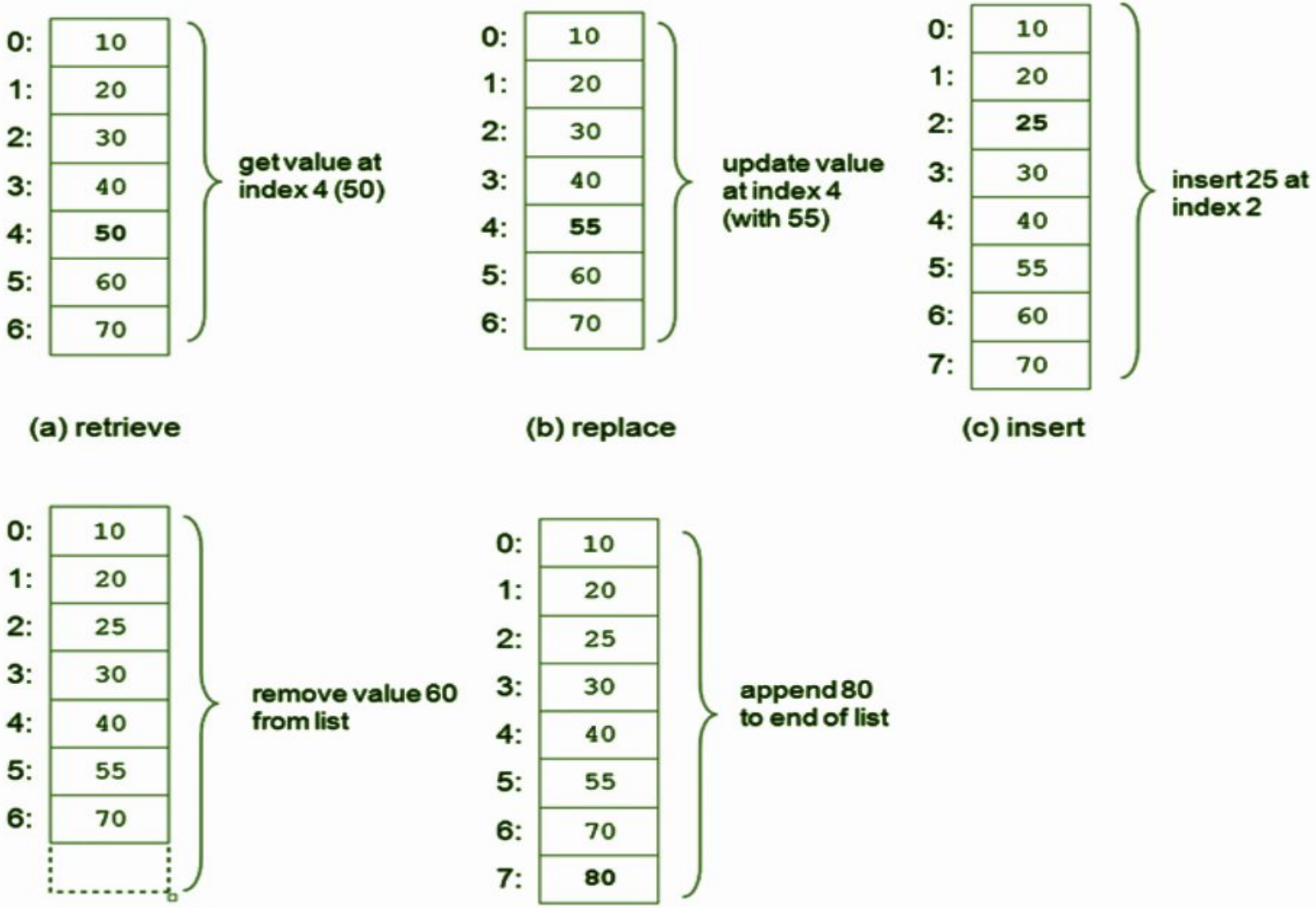
```
# List of Heterogenous Elements
MyListHeterogenous = [1,True, 4.5, 1+20j, 'PESUEC']
print(type(MyListHeterogenous))
print(MyListHeterogenous)
Length = len(MyListHeterogenous)
print('Length =', Length)
print(type(MyListHeterogenous[0]))
print(type(MyListHeterogenous[1]))
print(type(MyListHeterogenous[2]))
print(type(MyListHeterogenous[3]))
print(type(MyListHeterogenous[4]))
print(type(MyListHeterogenous[5]))
```

```
<class 'list'>
[1, True, 4.5, (1+20j), 'PESUEC']
Length = 5
<class 'int'>
<class 'bool'>
<class 'float'>
<class 'complex'>
<class 'str'>

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/tmp/ipykernel_7579/3575391736.py in <module>
     10 print(type(MyListHeterogenous[3]))
     11 print(type(MyListHeterogenous[4]))
---> 12 print(type(MyListHeterogenous[5]))

IndexError: list index out of range
```

pCPS 4.1.2 Common List Operations



Operations commonly performed on lists include **retrieve**, **update**, **insert**, **delete** (remove) and **append**.

pCPS 4.1.2 Common List Operations

```
# List of integer Elements
MyListIntegers = [10,20,30,40,50,60,70]
Length = len(MyListIntegers)
print(type(MyListIntegers))
print('Length ', Length)
print(MyListIntegers)

print('\nGet value at index 4')
print(MyListIntegers[4])

print('\nUpdate value at index 4 with 55')
MyListIntegers[4]=55
print(MyListIntegers)

print('\nInsert value 25 at index 2')
MyListIntegers.insert(2,25)
Length = len(MyListIntegers)
print('Length ', Length)
print(MyListIntegers)
```

```
<class 'list'>
Length  7
[10, 20, 30, 40, 50, 60, 70]

Get value at index 4
50

Update value at index 4 with 55
[10, 20, 30, 40, 55, 60, 70]

Insert value 25 at index 2
Length  8
[10, 20, 25, 30, 40, 55, 60, 70]

Remove value 60 from the list
Length  7
[10, 20, 25, 30, 40, 55, 70]
```

```
print('\nRemove value 10 from the list')
MyListIntegers = [10,20,10,20,10,20,10]
Length = len(MyListIntegers)
print(type(MyListIntegers))
print('Length ', Length)
print(MyListIntegers)
MyListIntegers.remove(10)
print(MyListIntegers)
```

```
Remove value 10 from the list
<class 'list'>
Length  7
[10, 20, 10, 20, 10, 20, 10]
[20, 10, 20, 10, 20, 10]
```

```
print('\nAppend 80 to end of the list')
MyListIntegers = [10,20,30,40,50,60,70]
Length = len(MyListIntegers)
print(type(MyListIntegers))
print('Length ', Length)
print(MyListIntegers)
MyListIntegers.append(80)
print(MyListIntegers)
```

```
Append 80 to end of the list
<class 'list'>
Length  7
[10, 20, 30, 40, 50, 60, 70]
[10, 20, 30, 40, 50, 60, 70, 80]
```

pCPS 4.1.2 Common List Operations

```
print('\nRemove an element based on the current index range of the list')
MyListIntegers = [10,20,30,40,50,60,70]
Length = len(MyListIntegers)
print(type(MyListIntegers))
print('Length ', Length)
print(MyListIntegers)
MyListIntegers.pop(0)
print(MyListIntegers)
print('Length',len(MyListIntegers))
MyListIntegers.pop(0)
print(MyListIntegers)
print('Length',len(MyListIntegers))
```

```
Remove an element based on the current index range of the list
<class 'list'>
Length  7
[10, 20, 30, 40, 50, 60, 70]
[20, 30, 40, 50, 60, 70]
Length 6
[30, 40, 50, 60, 70]
Length 5
```

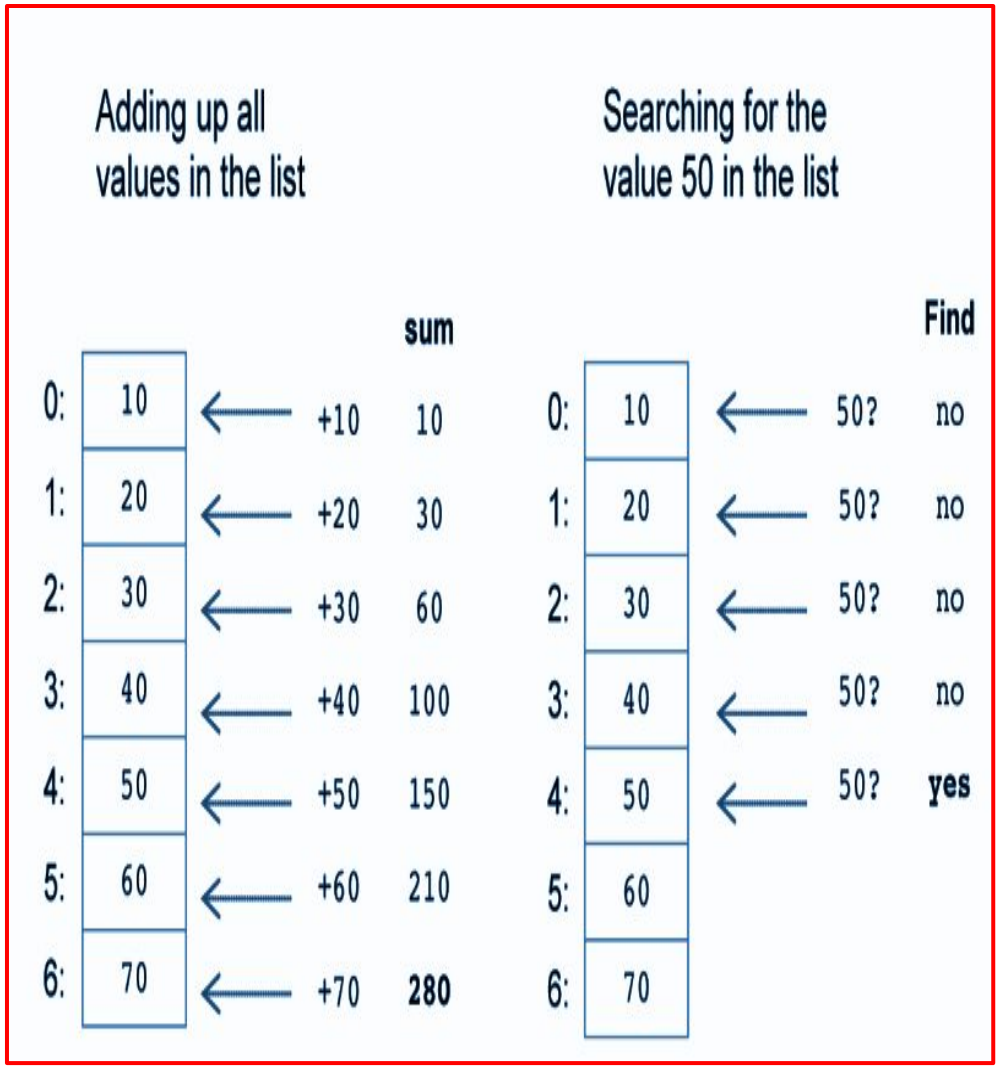
pCPS 4.1.2 Lists - Operations

List Characteristics	Elements
Element Type	All elements of the same type
	Elements of different types
Length	Fixed length
	Varying length
Modifiability	Mutable (alterable)
	Immutable (unalterable)
Common Operations	Determine if a list is empty
	Determine the length of a list
	Access (retrieve) elements of a list
	Insert elements into a list
	Replace elements of a list
	Delete elements of a list
	Append elements to (the end of) a list

List Properties and Common Operations

pCPS 4.1.3 List Traversal

- A **list traversal** is a means of **accessing elements**, **one-by-one**, of a **list**
- For example, to add up all the elements in a list of integers, each element can be accessed one-by-one, starting with the first, and ending with the last element.
- Similarly, the list could be traversed starting with the last element and ending with the first.
- To find a particular value in a list also requires traversal



pCPS 4.1.3 List Traversal

```
print('\Searching for a valid element list')
MyListIntegers = [10,20,30,40,50,60,50]
Length = len(MyListIntegers)
print(type(MyListIntegers))
print('Length ', Length)
print(MyListIntegers)
Index = MyListIntegers.index(50)
print('Index of 50 is',Index)
```

```
\Searching for a valid element list
<class 'list'>
Length  7
[10, 20, 30, 40, 50, 60, 50]
Index of 50 is 4
```

```
print('\Searching for a valid element list')
MyListIntegers = [10,20,30,40,50,60,50]
Length = len(MyListIntegers)
print(type(MyListIntegers))
print('Length ', Length)
print(MyListIntegers)
Index = MyListIntegers.index(500)
print('Index of 500 is',Index)
```

```
\Searching for a valid element list
<class 'list'>
Length  7
[10, 20, 30, 40, 50, 60, 50]

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_9541/228717997.py in <module>
      5 print('Length ', Length)
      6 print(MyListIntegers)
----> 7 Index = MyListIntegers.index(500)
      8 print('Index of 500 is',Index)

ValueError: 500 is not in list
```

pCPS 4.1.3 List Traversal

1. What would be the range of index values for a list of 10 elements?
(a) 0–9 **(b)** 0–10 **(c)** 1–10

2. Which one of the following is NOT a common operation on lists?
(a) access **(b)** replace **(c)** interleave **(d)** append **(e)** insert
(f) delete

3. Which of the following would be the resulting list after inserting the value 50 at index 2?

0:	35
1:	15
2:	45
3:	28

(a)

0:	35
1:	50
2:	15
3:	45
4:	28

(b)

0:	35
1:	15
2:	50
3:	45
4:	28

(c)

0:	50
1:	35
2:	15
3:	45
4:	28

PES UNIVERSITY ONLINE

This slide is intentionally left blank

pCPS 4.2.1 Python List Type

- A **list** in python is a **mutable**, **linear** data structure of variable length, allowing **mixed**-**type** elements.
- Mutable means that the contents of the list may be altered.
- Lists in Python use zero-based indexing.
- All lists have index values 0 ... n-1, where n is the number of elements in the list.
- **Lists** are denoted by a **comma**-**separated** list of elements **within** **square** **brackets**
- An **empty** list is denoted by an **empty** **pair** of **square** **brackets**, **[]**
- **Elements** of a list are **accessed** by using an **index** value within square brackets
- For **longer** **lists**, we would want to have a **more** concise way of **traversing** the elements
- **Methods** , and the associated **dot** notation used, are fully explained in later Chapter on **Objects** and their **Use**.
- **Methods** here for the **sake** of **completeness** in covering the topic of list operations

pCPS 4.2.1 Python List Type

Operation	fruit = ['banana', 'apple, 'cherry']	
Replace	fruit[2] = 'coconut'	['banana', 'apple', 'coconut']
Delete	del fruit[1]	['banana', 'cherry']
Insert	fruit.insert(2, 'pear')	['banana', 'apple', 'pear', 'cherry']
Append	fruit.append('peach')	['banana', 'apple', 'cherry', 'peach']
Sort	fruit.sort()	['apple', 'banana', 'cherry']
Reverse	fruit.reverse()	['cherry', 'banana', 'apple']

pCPS 4.2.1 Python List Type

```
print('\sorting the list ')
MyListIntegers = [10,20,30,40,50,60,50]
Length = len(MyListIntegers)
print(type(MyListIntegers))
print('Length ', Length)
print(MyListIntegers)
MyListIntegers.sort(reverse=True)
print(MyListIntegers)
```

```
\sorting the list
<class 'list'>
Length  7
[10, 20, 30, 40, 50, 60, 50]
[60, 50, 50, 40, 30, 20, 10]
```

```
print('sorting the Heterogenous list ')
MyListHeterogeneous = [1,2,3,4,'PESUEC']
Length = len(MyListHeterogeneous)
print(type(MyListHeterogeneous))
print('Length ', Length)
print(MyListHeterogeneous)
MyListHeterogeneous.sort()
print(MyListHeterogeneous)
```

```
sorting the Heterogenous list
<class 'list'>
Length  5
[1, 2, 3, 4, 'PESUEC']

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_9541/1294372305.py in <module>
      5 print('Length ', Length)
      6 print(MyListHeterogeneous)
----> 7 MyListHeterogeneous.sort()
      8 print(MyListHeterogeneous)

TypeError: '<' not supported between instances of 'str' and 'int'
```

pCPS 4.2.1 Python List Type

```
print('\sorting the list ')
MyListIntegers = [10,20,30,40,50,60,50]
Length = len(MyListIntegers)
print(type(MyListIntegers))
print('Length ', Length)
print(MyListIntegers)
MyListIntegers.sort(reverse=True)
print(MyListIntegers)
```

```
\sorting the list
<class 'list'>
Length  7
[10, 20, 30, 40, 50, 60, 50]
[60, 50, 50, 40, 30, 20, 10]
```

```
print('sorting the Heterogenous list ')
MyListHeterogeneous = [1,2,3,4,'PESUEC']
Length = len(MyListHeterogeneous)
print(type(MyListHeterogeneous))
print('Length ', Length)
print(MyListHeterogeneous)
MyListHeterogeneous.sort()
print(MyListHeterogeneous)
```

```
sorting the Heterogenous list
<class 'list'>
Length  5
[1, 2, 3, 4, 'PESUEC']

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_9541/1294372305.py in <module>
      5 print('Length ', Length)
      6 print(MyListHeterogeneous)
----> 7 MyListHeterogeneous.sort()
      8 print(MyListHeterogeneous)

TypeError: '<' not supported between instances of 'str' and 'int'
```

```
print('\Reversing the list ')
MyListHeterogeneous = [1,2,3,4,'PESUEC']
Length = len(MyListHeterogeneous)
print(type(MyListHeterogeneous))
print('Length ', Length)
print(MyListHeterogeneous)
MyListHeterogeneous.reverse()
print(MyListHeterogeneous)
```

```
\Reversing the list
<class 'list'>
Length  5
[1, 2, 3, 4, 'PESUEC']
['PESUEC', 4, 3, 2, 1]
```

pCPS 4.2.2 Python Tuples

- A **tuple** is an **immutable** **linear** data structure.
- Thus, in contrast to lists, once a **tuple** is defined, it **cannot** be **altered**.
- Otherwise, **tuples** and **lists** are **essentially** the **same**.
- To **distinguish** tuples from lists, **tuples** are denoted by **parentheses** instead of **square** **brackets**
- Another **difference** between **tuples** and **lists** is that **tuples** of **one** **element** **must** include a **comma** **following** the **element**
- An **empty** **tuple** is represented by a set of **empty** **parentheses**, ().
- The **elements** of tuples are **accessed** the **same** as **lists**, with **square** **brackets**
- Any attempt to **alter** a **tuple** is **invalid**.
- Thus, **delete**, **update**, **insert**, and **append** operations are not defined on **tuples**.
- For now, we can consider using tuples when the information to represent should not be altered.

pCPS 4.2.2 Python Tuples

```
# tuple of integer Elements
MyTupleIntegers = (1,2,3,4,5,0,-1,-2,-3,-4)
print(type(MyTupleIntegers))
print(MyTupleIntegers)
print('\n-------------------------------------')
print(MyTupleIntegers[4])
print(type(MyTupleIntegers[4]))
```

```
<class 'tuple'>
(1, 2, 3, 4, 5, 0, -1, -2, -3, -4)

-------------------------------------
5
<class 'int'>
```

```
# Tuple of float Elements
MyTupleFloat = (1.5,2.5,3.5,4.5)
print(type(MyTupleFloat))
print(MyTupleFloat)
print('-------------------------------------')
Length = len(MyTupleFloat)
print('Length =', Length)
print(MyTupleFloat[Length-1])
print(MyTupleFloat[-Length])
print(type(MyTupleFloat[-Length]))
```

```
<class 'tuple'>
(1.5, 2.5, 3.5, 4.5)
-------------------------------------
Length = 4
4.5
1.5
<class 'float'>
```

```
# Tuple of float Elements
MyTupleFloat = (1.5,2.5,3.5,4.5)
print(type(MyTupleFloat))
print(MyTupleFloat)
print('---------------------------------')
MyTupleFloat[3]=9.8
```

```
<class 'tuple'>
(1.5, 2.5, 3.5, 4.5)
---------------------------------

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_9541/2184632364.py in <module>
      4 print(MyTupleFloat)
      5 print('---------------------------------')
----> 6 MyTupleFloat[3]=9.8

TypeError: 'tuple' object does not support item assignment
```

pCPS 4.2.3 Sequences

- A **sequence** in python is a **linearly ordered** set of elements accessed by an **index** number.
- **Lists**, **tuples**, and **strings** are all **sequences**.
- **Strings**, like tuples, are **immutable** ; therefore, they **cannot** be **altered**.
- For **any sequence** s, **len**(s)gives its **length**, and **s[k]** retrieves the element at index **k**.
- The **slice** operation, s[index1:index2], returns a **subsequence** of a sequence, starting with the **index1** location up to but **not including** the **index2**.
- The **s[index :]** form of the slice operation returns a string containing **all** the list elements starting from the given index location to the end of the **sequence**.
- The **count** method returns how **many** instances of a given value occur within a sequence, and the **find method** returns the **index location** of the **first** occurrence of a specific item, returning -1 if not found.

pCPS 4.2.3 Sequences

```
MyList=[]
MyTuple = ()
MyString=''
print('MyList')
print(type(MyList))
print(len(MyList))
print('MyTuple')
print(type(MyTuple))
print(len(MyTuple))
print('MyString')
print(type(MyString))
print(len(MyString))
```

```
MyList
<class 'list'>
0
MyTuple
<class 'tuple'>
0
MyString
<class 'str'>
0
```

```
MyTuple = (1,2,3,4,5)
MyString=''
print('MyTuple')
print(type(MyTuple))
print(len(MyTuple))
MyTuple[2]=8.5
```

```
MyTuple
<class 'tuple'>
5

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_9541/3758864064.py in <module>
      4 print(type(MyTuple))
      5 print(len(MyTuple))
----> 6 MyTuple[2]=8.5

TypeError: 'tuple' object does not support item assignment
```

```
MyString='PESUEC'
print('MyString')
print(type(MyString))
print(len(MyString))
MyString[5] = 'E'
```

```
MyString
<class 'str'>
6

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_9541/1587665191.py in <module>
      3 print(type(MyString))
      4 print(len(MyString))
----> 5 MyString[5] = 'E'

TypeError: 'str' object does not support item assignment
```

pCPS 4.2.3 Sequences

```
print('Slice Operation')
MyList=[11,12,13,14,15,16]
print(len(MyList))
print(MyList)
print(MyList[0:10])
print(MyList[1:3])
print(MyList[0:6:2])
print(MyList[-1:-7:-1])
print(MyList)
```

```
Slice Operation
6
[11, 12, 13, 14, 15, 16]
[11, 12, 13, 14, 15, 16]
[12, 13]
[11, 13, 15]
[16, 15, 14, 13, 12, 11]
[11, 12, 13, 14, 15, 16]
```

```
print('Slice Operation on String')
MyString="PES University B.Tech First Semester P Section"
print(len(MyString))
print(MyString)
print(MyString[0:10])
print(MyString[1:3])
print(MyString[0:47:2])
print(MyString[-1:-47:-1])
print(MyString)
```

```
Slice Operation on String
46
PES University B.Tech First Semester P Section
PES Univer
ES
PSUiest .ehFrtSmse  eto
noitceS P retsemeS tsriF hceT.B ytisrevinU SEP
PES University B.Tech First Semester P Section
```

pCPS 4.2.3 Sequences

- For determining only if a given value occurs within a sequence, without needing to know where, the **in** operator can be used instead
- The **+** operator is also used to denote **concatenation**.
- Since the plus sign also denotes addition, python determines which operation to perform based on the operand types.
- Thus the plus sign, +, can be referred to as an **overloaded operator** .
- If both **operands** are **numeric** types, **addition** is performed.
- If both **operands** are **sequence** types, **concatenation** is performed.
- If a mix of numeric and sequence operands is used, an “unsupported operand type(s) for + ” error message will occur

pCPS 4.2.3 Sequences

```
MyList=[1,2,3,4]
MyTuple = (5,6,7,8)
MyString='PESU'

print(MyList+MyList)
print(MyTuple+MyTuple)
print(MyString+MyString)

print(MyList)
print(MyTuple)
print(MyString)
```

```
[1, 2, 3, 4, 1, 2, 3, 4]
(5, 6, 7, 8, 5, 6, 7, 8)
PESUPESU
[1, 2, 3, 4]
(5, 6, 7, 8)
PESU
```

```
MyList=[1,2,3,4]
MyTuple = (5,6,7,8)
MyString='PESU'
print(MyTuple+(0,))
```

```
(5, 6, 7, 8, 0)
PESUPESU
```

```
MyList=[1,2,3,4]
MyTuple = (5,6,7,8)
MyString='PESU'
print(MyTuple+MyList)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_9541/3392475991.py in <module>
      3 MyString='PESU'
      4
----> 5 print(MyTuple+MyList)

TypeError: can only concatenate tuple (not "list") to tuple
```

```
MyList=[1,2,3,4]
MyTuple = (5,6,7,8)
MyString='PESU'
print(MyList+MyTuple)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_9541/328611626.py in <module>
      3 MyString='PESU'
      4
----> 5 print(MyList+MyTuple)

TypeError: can only concatenate list (not "tuple") to list
```

pCPS 4.2.3 Sequences

- Operations min/max return the smallest/largest value of a sequence, and sum returns the sum of all the elements (when of numeric type).

- Finally, the comparison operator, == , returns True if the two sequences are the same length, and their corresponding elements are equal to each other.

pCPS 4.2.4 Nested Lists

- Operations min/max return the smallest/largest value of a sequence, and sum returns the sum of all the elements (when of numeric type).

- Finally, the comparison operator, == , returns True if the two sequences are the same length, and their corresponding elements are equal to each other.

- Lists and tuples can contain elements of any type, including other sequences.

- Thus, lists and tuples can be nested to create arbitrarily complex data structures.

pCPS 4.2.4 Nested Lists

```
MyList1=[1,2,3,4]
MyList2 = [1,2,3,4]
print(MyList1<MyList2)
print(MyList1==MyList2)
print(MyList1>MyList2)
```

```
False
True
False
```

```
MyList1=[81]
MyList2 = [11,12,3,4,91,92]
print(MyList1<MyList2)
print(MyList1==MyList2)
print(MyList1>MyList2)
```

```
False
False
True
```

```
MyList1=[11,12,13,14]
MyList2 = [1,2,3,4]
print(MyList1<MyList2)
print(MyList1==MyList2)
print(MyList1>MyList2)
```

```
False
False
True
```

```
MyList1=[11,12,13,14]
MyList2 = [1,2,3,80]
print(MyList1<MyList2)
print(MyList1==MyList2)
print(MyList1>MyList2)
```

```
False
False
True
```

pCPS 4.2.3 Sequences

Operation		String s = 'hello' w = '!'	Tuple s = (1,2,3,4) w = (5,6)	List s = [1,2,3,4] w = [5,6]
Length	`len(s)`	5	4	4
Select	`s[0]`	`'h'`	1	1
Slice	`s[1:4]` `s[1:]`	`'ell'` `'ello'`	`(2, 3, 4)` `(2, 3 ,4)`	`[2 ,3, 4]` `[2, 3, 4]`
Count	`s.count('e')` `s.count(4)`	1 *error*	0 1	0 1
Index	`s.index('e')` `s.index(3)`	1 --	-- 2	-- 2
Membership	`'h' in s`	True	False	False
Concatenation	`s + w`	`'hello!'`	`(1, 2, 3, 4, 5, 6)`	`[1, 2, 3, 4, 5, 6]`
Minimum Value	`min(s)`	`'e'`	1	1
Maximum Value	`max(s)`	`'o'`	4	4
Sum	`sum(s)`	*error*	10	10

pCPS 4.2.3 Sequences

```
import numpy as np
print('Some Lists Operations')
MyList=[11,12,13,14,15,16]
print(len(MyList))
print(MyList)
print(min(MyList))
print(max(MyList))
print(sum(MyList))
print(np.mean(MyList))
print(sum(MyList)/len(MyList))
```

```
Some Lists Operations
6
[11, 12, 13, 14, 15, 16]
11
16
81
13.5
13.5
```

```
MyList=[1,2,1,1]
MyTuple = (5,6,7,8)
MyString='PES UnivErsity PESU'

print(MyList.count(1))
print(MyTuple.count(1))
print(MyString.count('PES'))
print(MyString.count('E'))
print(MyString.count('I'))
print(MyString.upper().count('I'))
print(MyString.lower().count('p'))
```

```
3
0
2
3
0
2
2
```

THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University
nitin.pujari@pes.edu

For Course Digital Deliverables visit www.pesuacademy.com